A PARALLEL PROCESSING APPROACH TO SOFTWARE FAULT TOLERANCE

W. Kent Fuchs

Computer Systems Group
Coordinated Science Laboratory
University of Illinois
1101 W. Springfield Ave.
Urbana, IL 61801

(217) 333-9731

Abstract- The issue of concurrent detection and recovery from design errors in the software and physical failures in the hardware of parallel processor systems is considered in this paper. In contrast to classical *N-Version programming* and *recovery block* approaches to software fault tolerance, a specification-based approach to control and data structure verification is presented. The techniques use the hardware redundancy inherent in parallel processing systems to provide concurrent error detection and recovery.

Key Words- reliable software, concurrent error detection, fault tolerance

This research was supported by the National Aeronautics and Space Adminisration (NASA) under Contract NASA NAG 1-613 and by the Joint Services Electronics Program under Contract N00014-84-C-0149.

I. INTRODUCTION

There is an ever increasing need for high-performance reliable computation in many contexts of computer system application. In response to this need a large number of industrial and academic researchers have made significant contributions to the synthesis and analysis of techniques for enforcing fault-tolerant computing. Advances have been made both in the areas of hardware and software fault tolerance. However, there is a distinct lack of research concerning an integrated approach to software and hardware appropriate for parallel processing systems. Software fault tolerance has primarily consisted up to the present time of either the *N-version programming* or the *recovery block* approach. N-version programming is a method of enforcing design diversity and therefore error detection and recovery through N independently coded versions of a program [1]. The recovery block approach applies an acceptance test to a primary routine for purposes of error detection. A failure to pass the acceptance test results in a transfer of control to an alternate routine for attempted recomputation of the desired function [2].

Both of these techniques have been used to provide for toleration of both hardware and software errors in distributed environments [3, 4], while little concern has been given as to how software fault tolerance can be achieved in tightly-coupled parallel processing systems [5, 6]. Unfortunately, the application of N-version programming to hardware and software fault tolerance results in full replication of both hardware and software, while the recovery block technique necessitates the derivation of comprehensive acceptance tests, which is difficult for many computational tasks.

This paper introduces a specification-based approach to control and data structure verification which is appropriate for software and hardware fault tolerance in tightly coupled parallel processing systems. The techniques use the hardware redundancy inherent in a multiprocessor system to provide concurrent error detection and recovery. The focus of the

paper's contributions concern the concurrent detection of software design errors and hardware physical failures. Techniques for recovery concurrently under investigation also presented in summary.

II. RELIABLE PARALLEL COMPUTATION

A. Specification-Based Check Structures

The use of formal specifications is a significant aid in the detection of errors in the design of software. Several, formal specification languages have been developed specifically for software verification [7, 8, 1]. In this paper we propose the use of formal specifications to derive check structures which are used to concurrently monitor the validity of parallel software structures. The check structures function as redundant processes in the parallel computing environment whose control variables are concurrently compared with that of the executing nonredundant code. The nature of the objective in deriving the check structures is that they will provide a means of comprehensive error detection with significantly less development effort, performance penalty, and hardware allocation than classical fault tolerant schemes. Two specific examples of specification-based check structures are presented in summary below.

B. Control Structure Verification

The use of external monitors to detect errors in control flow have recently been developed and implemented [9, 10]. These techniques provide low-cost methods of hardware error detection. Our goal in this section of the paper will be to demonstrate that similar techniques can be used in parallel software to detect both design errors as well as hardware failures. Our approach is to use formal specification of the initial control, such as recently described by Lichtman [11], with concurrent comparison of the specified control structure with control variables stored in the shared memory of the parallel processor architecture. The initial formal specification provides for software error detection, while the use of distinct processors for

execution of the nonredundant parallel code and for comparison with the control check structure provides for detection of hardware failures.

C. Data Structure Verification

The integrity of data structures can also be provided for through a specification-based approach. Specification of abstract data types has been studied for purposes of program testing in recent literature [12]. Our research results incorporates data type specification with specification of valid data structures. Errors due to design and physical failure are detected by means of *access path verification* and comparison with the specification check structure.

III. CONCLUSION

The specification-based approach to control and data structure verification provides an alternative technique for software and hardware concurrent error detection in parallel processing environments. Current work concerns the implementation of recovery strategies from detected errors and the analysis of memory and performance costs for the proposed techniques.

REFERENCES

[1] A. Avizienis, "The N-Version Approach to Fault-Tolerant Software," *IEEE Trans. on Software Eng.*, vol. SE-11, pp. 1491-1501, December 1985.

[2] B. Randell, "System Structure for Fault Tolerance," *IEEE Trans. on Software Eng.*, vol. SE-1, pp. 220-232, Jun. 1975.

[3] S. V. Mukam and A. Avizienis, "An Event-Synchronized System Architecture for Integrated Hardware and Software Fault-Tolerance," *Proc. of the Fourth Inter. Conf. on Distributed Computing Systems*, pp. 357-365, May 1984.

[4] K. H. Kim, "Distributed Execution of Recovery Blocks: Approach to Uniform Treatment of Hardware and Software Faults," in *Proc. IEEE 4th Int. Conf. Distributed Comput. Syst.*, San Francisco, CA, pp. 526-532, May 14-18.

[5] K. H. Kim and C. V. Ramamoorthy, "Failure-Tolerant Parallel Programming and Its Supporting System Architecture," *Proc. of AFIPS*, pp. 413-423, 1976.

[6] Y. H. Lee and K. G. Shin, "Design and Evaluation of a Fault Tolerant Multiprocessor Using Hardware Recovery Blocks," *IEEE Trans. Comput.*, vol. C-33, pp. 113-124, Feb. 1984.

[7] J. V. Guttag and J. J. Norning, "An Introduction to the Larch Shared Language," in *Inform. Processing '83 Proc. IFIP Congr.*, Paris, France, pp. 809-814, Sept. 19-23, 1983.

[8] R. A. Kemmerer, "Testing Formal Specifications to Detect Design Errors," *IEEE Trans. on Software Eng.*, vol. SE-11, pp. 32-42, January 1985.

[9] J. P. Shen and M. A. Schuette, "On-Line Self-Monitoring Using Signatured Instruction Streams," *Proc. Int. Test Conf.*, pp. 275-282, Nov. 1983.

[10] M. Namjoo, "CERBERUS-16: An Architecture for a General Purpose Watchdog Processor," *Proc. Symp. Fault Tolerant Comput.*, pp. 17-20, 1983.

[11] Z. L. Lichtman, "Generation and Consistency Checking of Design and Program Structures," *IEEE Trans. on Software Eng.*, vol. SE-12, pp. 172-181, January 1986.

[12] I. J. Hayes, "Specification Directed Module Testing," *IEEE Trans. on Software Eng.*, vol. SE-12, pp. 124-133, January 1986.